TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kaspar Suurkaev 134560 IAPB

JAVA ISESEISVAKS ÕPPIMISEKS MÕELDUD TEEK

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kaspar Suurkaev

26.05.2019

Annotatsioon

Antud dokument annab ülevaate Java teegist, mille eesmärk on anda inimestele lisa viis kuidas iseseisvalt arendada oma programmeerimis oskusi Java programmeerimis keeles. See on disainitud selliselt, et kasutaja saaks enda poolt valitud arenduskeskkonnas iseseisvalt läbi töödelda mitmeid erinevaid programmeerimis ülesandeid. See teek ei nõua väliseid faktoreid, et kontrollida kasutaja lahenduse valiitsust, vaid kontrollib neid teegi enda sees.

Ülesanded annavad võimaluse selle teegi kasutajale lahendada mitmeid erinevas stiilis probleeme. Need võimaldavad kasutajal testida ja arendada oma oskusi Java programmeerimis valdkonnas iseseisvalt, saades tagasi sidet teegilt endalt oma lahenduse korrektsuse üle. See teek on eelkõige mõeldud inimestele, kes on suhteliselt uued Java programmeerimis keeles ning on ideaalis läbinud mõningad Java algajate kursused. Omades baas teadmisi progarmeerimisest, on kasutajal võimalik kasutada seda teeki, et arendada oma oskusi, eriti pannes rõhku sellele kuidas õppida ja ära harjuda kasutama Objekt Orineteeritud Programmeerimis põhitõdesid.

Lõputöö on kirjutatud Eesti keeles keeles ning sisaldab teksti 41 leheküljel, 36 peatükki, 4 joonist, 1 tabel.

Abstract

Programming Library for Independent Study of Java Language

This document will give an overview of a Java Library, which purpose is to give its user an additional way to study and learn Java programming language. It is meant to be an independent tool, requiring only the Library with a project that goes with it and an Integrated Development Environment (IDE) to work with. While many other solutions teach Java programming language through online coding in browser and tutorials, this is meant to be an offline tool, giving the user a chance to program and use IDE on their own, potentially without external help.

The exercises in this Library will allow the user to test their coding skills in multiple different ways. This Library is mostly meant for people who are quite new to the programming and have finished few basic courses in Java programming. With the basic knowledge of programming, this Library should help them to learn more about little bit more complex parts of the Java programming language, especially how to start using Object Orientated Programming (OOP) more optimally and make use of the benefits that OOP gives them.

By the end of it, if user has finished all exercises given in this Library, they should have more experience in how to use OOP and feel more comfortable with working with more than just most basics of the Java programming language.

The thesis is in Estonian and contains 41 pages of text, 36 chapters, 4 figures, 1 table.

Lühendite ja mõistete sõnastik

Access-modifier	Ligipääsu piiraja
Constructor	Konstruktor – meetod, mida kutsutakse klassi loomisel välja
enum	Enumeraator – klassi tüüp, mis sisaldab ainult konstant muutujaid.
Getter	Meetod, mis tagastab välja väärtuse.
IDE	Integreeritud arenduskeskkond, inglise keeles *Integrated Development Environment*
Interface	Liides
JOOP	Java Objekt-Orienteeritud Programmeerimine
main method	Java-s spetsiaalne meetod, mis tähistab koodi kõige esmast jooksutamist kohta, kust kood teab alata.
Object First Programming	Objekt enne programmeerimine – õpetamise viis, kus koos lihtsamate programmeerimisvõttetega õpetatakse koos ja objekte tundma, haldama, ja kasutama.
Package-private	Pakketi-privaatne
Programming First	Programmeerimine enne – õpetamise viis, kus esmalt õpetatakse lihtsaid programmeerimisvõtteid ja alles seejärel objekte.
Puzzle	Pusle – viitab pusle probleemi lahendus metoodikale.
Reflection	Peegeldus – metoodika, mille abil saab kood ennast uurida ja analüüsida koodi jooksutamise käigus
Teek	Inglise keeles *Library*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Lõputöö eesmärk on luua inimeste jaoks, kes on endale selgeks teinud Java programmeerimiskeele põhitõed ja soovivad jätkata oma oskuste lihvimist, Java *library* (teek) koos sellega kaasas käiva projektiga. See projekt ja sellega kaasas käiv *library* annaks võimaluse Java keele õppijale lisada oma õppevahendite hulka iseseisvalt lahendamiseks mõeldud toote.

Siin töös käsitletav projekt annab selle kasutajale võimaluse iseseisvas korras ilma väljastpoolt tuleva tagasisideta arendada oma Java programmeerimisoskusi. Projekt annab selleks kasutajale 6 ülesannet. Need ülesanded on ehitatud üles kahe erineva disaini peal. Esimene disainiidee on proovida kasutajale selgitada ja valgustada teatud Java Objekt-Orienteeritud Programmeerimise (JOOP) põhitõdede kohta. Teine idee on laiemas perspektiivis anda kasutajale üldisem probleem lahendada, mille lahendus hõlmaks erinevate JOOP ja edasijõudnumate lahenduste kasutamist. Tähtsal kohal on kasutajas tekitada suurem mugavustunne kasutada, hallata ja luua ise objekte. See oskus aitab kaasa tulevastes programmeerimisülesannetes, kuna iga vähegi suurem tarkvara projekt kasutab tihedalt JOOP-e kasutamist.

Omandades neid oskusi ja saades juurde teadmisi programmeerimises, tulevad meie tööturule suurema kogemustepagasiga ning oskuslikumad programmeerijad. See on samuti aspekt, mida see lõputöö soovib saavutada oma lõpplahenduse pakkumisega kasutajatele.

1.1 Teegi loomis metoodika

Selles lõputöös on vaja luua teek, mis omaks endas ülesandeid kasutajatele. Selle jaoks, et seda teeki luua on, kasutatakse Java 8-t. See on piisavalt uus versioon sellest programmeerimiskeelest, et omada häid ja kasulike funktsionaalsusi, kuid mitte liiga vana ega liiga uus, et see ei näeks programmeerimise tööturul kasutust.

Esmane eesmärk on välja mõelda ja disainida ülesannete ideed. Need ideed peaksid olema reaalelulise väljundiga, et kasutajal oleks lihtsam seostada oma koodi reaalse maailmaga. Selliselt on kasutajal lihtsam aru saada talle etteantud ülesannete eesmärkidest ning see aitab tal paremini mõista, kuidas kasutada JOOP põhimõtteid.

Kui ideed on valmis, siis järgmisena hakatakse looma neid väljamõeldud ülesandeid. Selle jaoks tehakse valmis teegi baasstruktuur ja loogika, mis lubaksid selle teegi struktuuri sisse lisada uusi ülesannetega seotud objekte. Niimoodi on hea alusstruktuur olemas, mis lubab kasvada projektil ilma, et hiljem peaks suuri muudatusi kogu projekti peal tegema.

Peale seda, kui ülesanded on loodud ning teek on olemas, luuakse projekt, kuhu sisse lisatakse lihtne ja algeline struktuur ülesannete jaoks. See struktuur on mõeldud kasutajale sisendi ja väljundi korrektseks vormistamiseks. Selle projekti hulgas on ka olemas selle jaoks loodud teek ning dokumentatsioon, mida oleks võimalik läbi veebilehitseja või läbi oma arenduskeskkonna kasutada.

Kui kõik need osad on loodud, siis on selle lõputöö praktiline osa tehtud.

1.2 Lõpptulemus

Lõpptulemusena peab valmima esmalt teek, mis omab endas kõike vajalikku ülesannete etteandmiseks, lahendamiseks ja kontrollimiseks. Samuti peab eksisteerima ka projekt, mida oleks võimalik kasutajatele edasi anda, et nad saaksid seda loodud teeki kasutada ja enda oskusi arendada. Selles avalikus projektis peab olema ka dokumentatsioon, mida kasutaja saaks kasutada, et saada täpsemat ülevaadet talle kättesaadavatest objektidest ja nende sisust, mida tal ülesannete lahendamisel vaja on.

2 Programmeerimise õpetamine

2.1 Projekti õpetusmeetodid

Tänapäevases tehnoloogilises keskkonnas on aina kasvav vajadus tarkvara arendajate järgi [1] [2] . Kuna nõudlus on kõvasti suurem kui saadavus, siis on aina tähtsamal kohal motiveerida inimesi tulema ja õppima, kuidas programmeerida ja tarkvara lahendusi luua. Seetõttu on tähtsal kohal ka leida viise, kuidas hästi inimestele seda valdkonda tutvustada ja neile selgeks teha programmeerimise põhitõdesid.

2.1.1 „Objekt esmalt programmeerimine" metoodika kasutus

Kaks laialdasema definitsiooniga programmeerimise õpetamise viisi on *Programming First* (Programmeerimine esmalt) ning *Object First Programming* (objekt esmalt programmeerimine) [4] . Need on kaks erinevat õpetusviisi, kus esimeses õpetatakse alguses ainult programmeerimise algteadmisi ja siis minnakse edasi keerulisemate kontseptsioonide peale [4] . Teine õpetusviis võtab aga prioriteediks koos algteadmistega õpetada ka samal ajal objektide kasutamist.

Kuigi *Object First* meetoodika on mõnevõrra keerulisem viis alustada õpimist [3] , siis on sellel siiski omad eelised ning sellega metoodika ja OOP-st arusaamise jaoks on tehtud erinevaid uurimusi[4] [5] . Nendes on välja toodud klassidest ja objektidest arusaamise tähtsus ning põhjendatud selle teadmise vajalikkust programmeerijale. Tänapäevases tarkvaraarenduse sektoris on väga keeruline ilma OOP-d mõistmata hakkama saada ning seetõttu on ka *Object First* metoodika näidanud aina suuremat populaarsuse kasvu.

See lõputöö põhineb osaliselt *Object First* lähenemisel. Kuna siin lõputöös saavutatav eesmärk on anda kasutajatele lisa õppevahend, mis hõlmab suures koguses objektidega suhtlust, siis saab väita, et on kasutusel mitmeid *Object First* metoodika lähenemisi. Samas ei saa seda lugeda täielikult *Object First* lähenemist jälgivaks lõpptulemiks, sest

selle teegi kasutusel on tehtud eeldus, et kasutajad on juba teadlikud baas-programmeerimisest ning neil on esamane arusaam objektidega töötamisest. See eeldus on võimalik saavutada nii *Programming First* kui ka *Object First* meetoodikaga, mistõttu ei saa ka öelda, et see teek tugineks ja eeldaks täielikult *Object First* metoodikat, vaid ainult osaliselt.

Samas on siin teegis loodud ülesannete keerukus piisavalt madal, et seda saaks kasutada *Object First* õppeprogrammis, peale esmaseid selle õppeviisi kursusi läbides ilma, et kasutajatel oleks liiga suured puudujäägid siinsete ülesannete lahendamiseks.

2.1.2 „Probleemi lahendamine" ja „pusle" meetodika

Selle teegi üks suur osa on proovida õpetada inimestele, kuidas paremini programmeerida. Selleks on valitud kaks metoodikat. Enamus ülesandeid kasutavad probleemi-põhist lähenemist ning üks ülesanne kasutab *Puzzle* (Pusle-põhist) lähenemist [6] .

Probleemipõhine metoodika annab võimaluse kasutajale ette anda ülesande, millel oleks teatud reaalne murepunkt, mida võiks ka igapäevases elus tarkvara arendaja tööelus ette tulla. Andes kasutajale ette ülesande potentsiaalselt realistliku murega, aitab see tal paremini seostada, kuidas erinevaid probleeme lahendada nii talle ette antud ülesandes kui ka tulevikus teistes programmeerimispõhistes probleemides.

Puslepõhine metoodika tähendab eelkõige panna kasutaja mõtlema väljaspool nende potentsiaalset mugavustsooni. See annab kasutajale ette üldise probleemi, aga mitte kohese lähenemistee, kuidas probleem lahendada. Sellises ülesandes on vaja kasutajal hakata ise uurima täpsemalt, mida teatud ülesandes esinevad komponendid teevad ning proovida nende abil leida lahendus etteantud probleemile. Lahendus ei pruugi olla koheselt arusaadav, mistõttu on võimalik, et kasutaja peab mitut eri lahendust proovima enne kui ta õige leiab. See aga aitab arendada kasutaja loovat mõtlemist, mis omakorda arendab kasutaja probleemi lahendamise oskust väljaspool tavalist lähenemist, sest alati ei ole võimalik arendajale etteantud probleeme lahendada juba tuttavate lahendustega, vaid on vaja leida mingi uus ja seni kasutama lahendus.

2.2 Sarnased õppe-lahendused

Java õppimiseks on olemas mitmeid erinevaid lahendusi. Inimestel on võimalus õppida ülikoolis ning omandada teadmisi Java kohta. Samuti on võimalik targemaks saada gümnaasiumis, kus on aina rohkem hakatud õpetama programmeerimist.

Üks väga suur osa õppeprotsessist on aga iseseisev õppimine. Selleks on olemas suurel hulgal raamatuid, mis õpetavad inimestele nii algelisi kui ka endasijõudnute teadmisi Java programmeerimise keeles.

Kuid kõige suuremaks võrdluseks seoses selle teegiga ma tooks välja digitaalsed õppematerjalid. Tänapäevaks on internetis suurtes kogustes materjali, mida on võimalik leida ja kasutada teadmiste omandamiseks. On olemas videomaterjalid *Youtube*'s[1]. Samuti on suures koguses erinevaid veebilehti, mis on ennast spetsialiseerinud programmeerimise ja sealhulgas Java keele õpetamisele läbi teoreetilise info jagamise[7] [8] [9] [10] .

Ühed kõige ligilähedasemad tooted minu lahendusele on veebilehitsejapõhised lahendused, kus on võimalik reaalset koodi kirjutada ja seda testida otse veebilehitsejas[10] [11] [12] .

Samuti on olemas õppetarkavara nimega BlueJ [2], mis kasutab visualiseeritud lähenemist õpetamaks kasutajale JOOP arusaama. Selle tarkvara eesmärk on osaliselt sarnane käesolevas lõputöös loodud teegi ja kasutajale antava projektiga, kuid nende kahe toote implementatsioon õpetamiseks on erinev.

BlueJ õpetab mõnevõrra abstraktsemal tasemel JOOP kasutama, andes kasutajale visuaalselt sõbralikuma lahenduse kuidas objekte luua ja neid seostada. See on hea viis, kuidas esimesi kontseptsioone ja arusaamu õpetada JOOP kohta.

Selle lõputöö kasutajale antav lõpplahendus annab samuti võimaluse kasutajal JOOP teadmisi arendada, kuid erinevus seisneb selles, et see lõputöö kasutab igapäevaselt kasutatavaid tööriistu ja vahendeid. Nii õppides on kasutajal võimalik saada reaalset

1 https://www.youtube.com/

2 https://www.bluej.org/

arenduskogemust, kuidas projektidega, mis eeldavad JOOP teadmiste kasutamist, tööd teha.

2.3 Käesoleva lõputöö kasulikkusja erinevus teistest õppelahendustest

Eelmises peatükis on nimekiri erinevatest vahenditest, mille kaudu või mida kasutades saaks kasutaja uusi teadmisi omandada ja rakendada. Enamus neist nõuavad interneti kaudu informatsiooni kättesaamist ja sinna selle laadimist (veebilehitseja põhiste ülesannete lahendamine) või õpetavad abstraktsema lähenemisega, kuidas JOOP-d kasutada.

Selle lõputöö struktuur ja kasutamine on mõnevõrra teistsuguse lahendusega. Siin välja toodud lahendus on mõeldud kasutamiseks kasutaja enda arenduskeskkonnas, enda masinas ja ise seda koodi läbi jooksutades. Otsides samasuguseid lahendusi, ei tõusnud esile ühtegi populaarset lahendust, mis on laialdaselt tuntud või tunnustatud.

Seetõttu otsustasin luua teegi, mis omaks endas mitmeid ülesandeid, funktsionaalsust (mis oskaks kontrollida ja valideerida kasutaja lahendust) ning anda selle põhjal lihtne tagasisise lahenduse korrektsuse kohta. Selle teegiga käib kaasas ka väike projekt, mis omab endas iga ülesande jaoks eraldi Java faili ning eraldi *main method* faili, kus määratletakse ära, mis ülesannet testima hakatakse. Täpsem kirjeldus kuidas teek ja projekt töötavad, on välja toodud 3. peatükis.

Käesoleva lõputöö kasulikkus on järgnev: kasutades teeki ja sellega kaasaskäivat projekti, antakse kasutajale juurde kas lisa- või alternatiivlahendus, kuidas oma oskuste pagasit täiendada. Kasutades lahendust, antakse kasutajale võimalus oma enda *IDE-s* (Integrated Development Environment, integreeritud arenduskeskkond) tööd teha juba olemasolevate vahenditega. Kuigi alati on teretulnud iseseisvalt projekti loomine ja mõne enda idee realiseerimine, siis paljudele inimestele võib siinne lahendus olla lihtsam, millega algust teha peale Java baasteadmiste omandamist.

Kindlasti on tähtis rõhutada, et see lahendus ei ole mõeldud asendamaks teisi õppevahendeid ja õpetamismeetodeid, vaid seda tuleks vaadata kui lisavahendit neile, kes soovivad enesearenguga tegeleda minimaalselt vajalikust ja nõutust enam. Samuti

toon välja, et kogenud arendajatele ei ole siinsed ülesanded keerukad ning see ei ole toode, mis oleks ka neile mõeldud. Kõik on teretulnud seda toodet kasutama, aga kindlasti ei ole see kõikehõlmav lahendus Java programmeerimiskeele õppimiseks.

3 Lõputöö teegi ja projekti struktuur ning ülesehitus

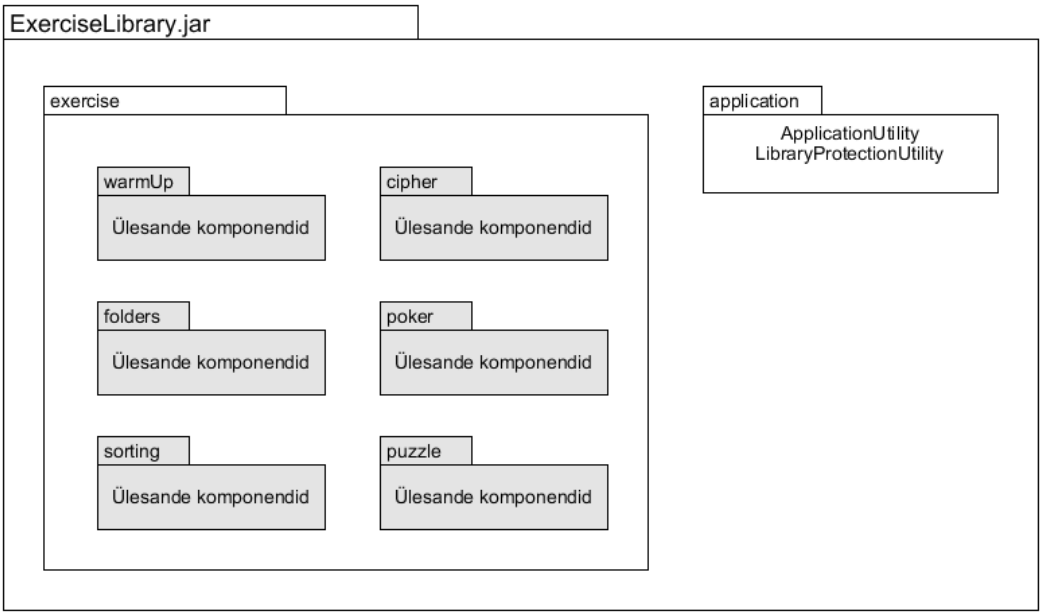
Siin peatükis selgitatakse täpsemalt ära, kuidas on struktureeritud nii teek kui selle jaoks mõeldud projekt, kuidas nende töövoog toimib ning mida on tehtud sohivastaseks kaitseks.

Käesolev lõputöö projekt jaotub kaheks osaks. Esimene on teek ja selle projekt, mis omab endas ülesannete objekte, nende lahendamise loogikat ning kasutaja sisendite kontrolli.

Teine osa on kasutajale antav projekt, mis omab algelist struktuuri kasutajale mugavalt ning ilma suure vaevata ülesannete lahendamise alustamiseks. See kasutaja projekt omab juba lõputöö jaoks loodud teeki ning kasutab ära seal sees olevaid komponente, et kasutajale ette anda ülesandeid ja nende sisendeid.

3.1 Teegi struktuur

Selle lõputöö jaoks loodud teegi baas-struktuur on välja toodud järgneval joonisel.



Joonis 1. Teegi lihtsustatud struktuur

Teek omab nime `ExerciseLibrary.jar`. Teegi sees on kaks kausta: `exercise` ja `application`.

Kaustas `exercise` hoitakse ülesannete komponente, mida kasutaja saab kasutada ülesannete lahendamise alustamiseks. Nende komponentide hulka kuuluvad ülesandeks vajalikud klassid, iga ülesande *Interface* (Liides) ning ülesande testimiseks vajalikud klassid. Täpsema ülevaate iga ülesande klassidest ja sisust on välja toodud peatükis 4.

Kaustas `application` hoitakse kahte faili: `ApplicationUtility` ja `LibraryProtectionUtility`

- `ApplicationUtility` - See klass on loodud selleks, et kasutaja saaks seda kasutada oma lõpplahenduse ülesande määrajana. Seal on olemas üks ainus

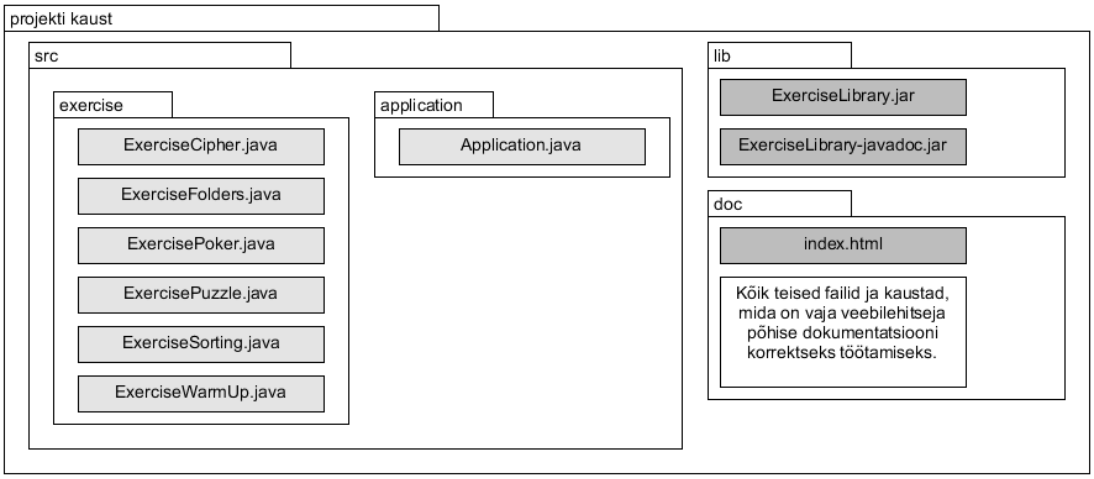
staatiline meetod nimega `runExercise`. See meetod võtab sisendina `Object` tüüpi muutuja ning kasutab seda, et määrata ära, millist ülesannet hakatakse kontrollima.

- `LibraryProtectionUtility` – See on loodud selleks, et erinevates ülesannetes ja testimisklassides oleks võimalik lisada meetoditele juurde lisa kaitsekiht, mis aitaks vältida *reflection* (Peegeldus) tüüpi privaatsete meetodite kätte saamist. Mitmed meetodid on ülesannetes mõeldud kasutamaks ainult teegi enda poolt ning selle jaoks, et tagada selle reegli püsivus, on lisatud mitmetesse meetoditesse `LibraryProtectionUtility` meetod `isMethodCalledFromRightPlace`, mis kontrollib välja kutsutud meetodi asukoha valiidsust.

  Sellest on täpsem informatsioon saadaval peatükis 3.4.

3.2 Projekti Struktuur

Kasutaja jaoks mõeldud projekti esmane struktuur on visualiseeritud järgneval joonisel.



Joonis 2. Kasutaja projekti esialgne struktuur

Kasutaja alustab seda projekti nelja põhilise osaga:

1. Projektis on olemas vajaminev teek, mis omab endas kõike vajaminevaid klasse, mida kasutatakse ülesannetes. Samuti on seal ülesannete lahenduse kontrollimiseks vajalik loogika ning muud valideerimis- ja sohitegemise vastased kaitsemeetodid.

2. `Application.java` fail, mida kasutatakse projekti jooksutamise alguspunktina. Sellest on täpsemalt kirjeldatud peatükis 3.3

3. Ülesannete java failid. Need failid implementeerivad erinevaid ülesannete liideseid. Need liidesed lubavad projektis oleval teegil lõpputulemusena jooksutada teste kasutaja lahenduse vastu.

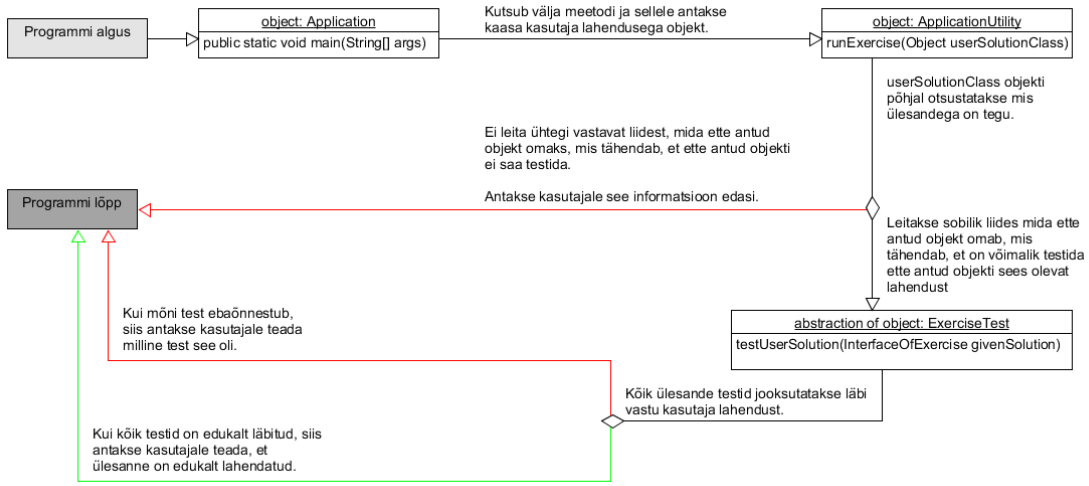
4. Veebilehitseja kaudu kasutatav dokumentatsioon, mis lubab kasutajal saada parema ülevaate tema jaoks olemasolevatest ülesannete lahendamiseks vajaminevatest ressurssidest. Samuti on seal ära toodud iga ülesande eesmärk

vastavate failide all. Seda dokumentatsiooni saab vaadata, kui avada `doc` kasutas olev fail `index.html`

Iga ülesande fail omab endas üht või rohkemat meetodit, kuhu kasutaja peaks oma lõpp-lahenduse tagastama. Igal meetodil on samuti olemas sisendandmed, mille põhjal kasutaja saab hakata ülesannet lahendama.

3.3 Koodi töötamise töövoog

Järgnev joonis toob välja lihtsustatud kujul, kuidas töötab kasutaja arvutis jooksutatav projekt, kui see tööle panna.



Joonis 3. Projekti töövoog, kui kasutaja selle käivitab

Selleks, et kasutaja saaks programmi jooksutada, on esmalt vajalik, et oleks täidetud järgnevad nõuded ja soovitused:

- Kasutaja arvutis peab olema Java 8 või uuem versioon, mis suudaks nii kompileerida projekti kui ka jooksutada kompileeritud koodi.
- Kasutaja on enda kätte saanud projekti failid, mida soovitavalt kasutada mõne arenduskeskonnaga (nagu näiteks Intellij, Eclipse vms).

- Kasutaja omab baasteadmisi, kuidas Java koodi kirjutada ning kuidas projekte hallata ja neid jooksutada.

Kui kasutaja on eeltööga ühel pool, saab alustada mõne ülesande lahendamisega ning kui ta on jõudnud piisavalt kaugele testimaks oma lahenduse sobivust, on tal vaja anda `ApplicationUtility.runExercise` meetodile ette ülesande lahenduse objekt ja see käivitada.

Seejärel juhtub tehnilisel tasemel järgnev:

Töö voog algab `main` meetodist, kus peaks ideaalis olema ainult üks käsk. See käsk peaks olema:

```
ApplicationUtility.runExercise(Object userSolutionClass);
```

Sellele meetodile peaks kasutaja andma väärtuseks ühe eelnevalt projektis esineva ülesande klassi, kus on juba vajalikud meetodid olemas ning mille klass implementeerib õiget liidest. Selles klassis peaks ettenähtud meetoditele olema kätte antud ülesande lahendus, mis lubaks projektiga kaasaskäival teegil kasutaja lahendust kontrollida.

Olles `ApplicationUtility.runExercise` meetodi välja kutsunud, tegeleb see meetod kasutaja sisendi analüüsiga. Lühendatud kujul, kuid meetodi põhiloogikat välja toov kood on järgnev:

```
private static final String ERROR_INCORRECT_INTERFACE = "ERROR: Your exercise
does not implement the correct interface!";

public static void runExercise(Object userSolutionClass) {
	if(userSolutionClass instanceof WarmUpInterface) {
		WarmUpInterface solution = (WarmUpInterface) userSolutionClass;
		WarmUpTests.testUserSolution(solution);
	}
	/* ...
	 * Siin on kõik teised else if koodid jupid, mis kontrollivad sama
	 * moodi userSolutionClass muutuja sobivust teiste liidestega.
	 */ ...
	else System.out.println(ERROR_INCORRECT_INTERFACE);

}
```

See meetod kontrollib kasutaja poolt ette antud objekti vastavust projektis esinevate ülesannete liidestega. Kui vastavus on olemas, siis on teada, et kasutaja objekt omab õigeid meetodeid, mida saab kasutada testideks. Seepeale saab kood oma tööd jätkata. Kasutaja sisend konverteeritakse ümber sobivasse struktuuri, et teek saaks kasutada sobilike meetodeid ning seejärel kutsutakse välja liidesele vastav testi klassi meetod ning antakse kasutaja objekt sellele meetodile kaasa.

Kui aga kasutaja objekt ei oma ühtegi õiget liidest, siis ei ole võimalik teegil kasutaja lahenduse valiidsust kontrollida ning kasutajale kuvatakse seejärel teade, et midagi on valesti. See annab kasutajale teada, et ta peab üle kontrollima, kas ta on ette antud klassides teinud liidese implementatsiooniga mingeid sobimatuid muudatusi või ei anna ta lihtsalt õiget sobiliku sisendit ette.

Eeldusel, et kõik töötab sinnamaani, minnakse edasi testklassis kasutaja lahenduse testimisega. Seal on vastavalt ülesandele olemas erinevad testid, mis kontrollivad kasutaja lahenduse korrektsust. Kui kõik õnnestub, siis kuvatakse teade selle kohta ning programm lõpetab oma töö.

Kui üks või rohkem testidest ebaõnnestub, siis antakse antakse kasutajale teada nende testide ebaõnnestumisest ning kuvatakse kasutajale ka ebaõnnestunud testide koguarv.

3.4 Sohitegemise elementaarne kaitse

Kuna kasutaja peab teegi osasid tihedalt kasutama, on vaja tagada esmane kaitse selle vastu, et kasutajad ei saaks lihtsalt niisama kätte infot, mis ei ole neile lahendamise kasutamiseks mõeldud. Selliste asjade alla kuuluvad näiteks meetodid, mida kasutatakse testimises saamaks kätte õigeid lahenduste tulemusi; muutujad, mis on mõeldud ainult teegi siseseks kasutuseks ning meetodid, mis seavad muutjatele väärtusi, aga mida kasutaja ei tohiks kätte saada ja kasutada.

Selleks, et piirata selliste ja teiste vajalike komponentide kättesaadavust, on teegis rakendatud paari eri lahendust.

3.4.1 Juurdepääsu piirajad

Esimene viis, kuidas kaitsta komponente kasutajale kättesaadavuse eest on Java keele enda keelepõhised nüansid. Nimelt on võimalik kasutada *access modifier* (juurdepääsu piirajad), mis lubavad erinevatele klassidele, muutujatele ja meetoditele lisada ette võtmesõnasid. Need võtmesõnad piiravad õigusi, kust kood võib komponentidele ligi saada.

Javas olemasolevad juurdepääsu piirajad on: `public` (avalik), `protected` (kaitstud) ja `private` (privaatne). Samuti on olemas selline tase nagu *package-private* (pakki-privaatne), aga selle jaoks ei ole eraldi võtmesõna kasutusel [13] .

Allolev tabel annab kiire ülevaate, mis tasemetel mainitud võtmesõnad piiranguid kehtestavad:

Tabel 1. Juurdepääsu võtmesõnade piiramise tasemed

Võtmesõna	Klass	Pakk	Alamklass	Üle projekti
public	Jah	Jah	Jah	Jah
protected	Jah	Jah	Jah	Ei
Ilma võtmesõnata/ package-private	Jah	Jah	Ei	Ei
private	Jah	Ei	Ei	Ei

`public` võtmesõna lubab selliselt defineeritud komponentidele saada ligi ja neid välja kutsuda ja luua igalt poolt. See on kõige avatum tase ning kõik komponendid, mida kasutaja peab saama kasutada on defineeritud `public` võtmesõnaga.

`protected` võtmesõna lubab ligi saada komponentidel, mis on kas alamklassid, paki sees olevad klassid või klass ise. Selliselt on teegis defineeritud mitmed komponendid, mis on vajalikud testimise jaoks kättesaadavaks teha, aga mitte kasutajale. Nende hulka kuuluvad näiteks meetodid, mis leiavad teegisiseselt õige lahenduse ülesandele, et seda võrrelda kasutaja lahendusega, teatud muutujate ligipääsud ning mõned *constructor* (klassi loomisel välja kutsutav meetod, konstruktor eesti keeles) meetodid.

Package-private lahendus on mõeldud enamasti klassi tasemel piirangu defineerimiseks, sest muutujad ja meetodid võiksid korrektsuse mõttes alati piirangu

võtmesõna omada. Kui klass on defineeritud ilma piirangu võtmesõnata, siis töötab see peaaegu samamoodi kui `protected` võtmesõna – kõik omadused on samad, mis `protected`, ainult et klass ei saa omada sellist defineerivat võtmesõna, mistõttu ilma võtmesõnata omistatakse klassile selline piirang. See lahendus on kasutusel just sellistele klassidele, mida on vaja ainult teegi enda kasutuseks, aga mida ei tohiks kasutada saada kasutaja ise.

`private` võtmesõna on kõige suurema piiranguga võtmesõna. See annab ainult klassi-sisestele komponentidele ligipääsu. See on väga levinud võtmesõna klassisisestele muutujatele, mida ei soovita otseselt välja anda teistele koodi osadele. Sama põhjus on ka siis teegis. Mitmed muutujad ja meetodid on mõeldud vaid klassisiseseks kasutamiseks ning need ei ole mõeldud, et klassist väljaspoolt keegi neid kasutaks.

3.4.2 *Reflection* metoodika vastane baaskaitse

Reflection metoodika on Java-s funktsionaalsus, mis lubab koodil ise ennast ja teisi koodi osasid uurida ja mõjutada koodi jooksmise hetkel [14] [15] . See on omapärane funktsionaalsus, mis ei leia nii tihedalt kasutust, kui teised Java keele osad, sest üldjoontes võiks kood olla selliselt disainitud, et koodi jooksmise ajal ei oleks vaja koodi ennast analüüsida ja muuta.

Kuid kuna selline võimalus on Java-s olemas, siis on ka potentsiaalne oht, et kasutaja võib seda kasutada koodi sees informatsiooni või funktsionaalsuste kättesaamiseks, mis pole talle mõeldud. Siinkohal tuleb välja tuua, et see on mõeldud eelkõige vältimaks selliseid juhtusid, kus kasutaja on midagi sellisest asjast kuulnud, kuid ei hooma täielikult selle tähendust ning siis proovib mõne internetist või mujalt leitud koodiga saavutada piiratud komponentidele ligipääsu saamist.

Inimene, kes juba mõistab, mis *reflection* on ja kuidas seda efektiivselt kasutada, peaks enamasti olema inimene, kelle programmeerimise teadmised on rohkemad, kui see, kelle jaoks see projekt mõeldud on. See kaitse ei ole mõeldud nende takistamiseks, sest nemad ei ole selle projekti sihtgrupp ning kui nad on suutelised ja soovivad sohki teha, siis see on nende enda otsus.

Kuid teadmatusest tulenev või mittetäielikult *reflection* kontseptsiooni hoomava kasutaja sohivastaseks kaitseks on teegis olemas selline klass nagu `LibraryProtectionUtility`. Sellel klassis on üks ainuke meetod `isMethodCalledFromRightPlace(List<String> allowedClasses)`.

Selle meetodi kood on järgnev:

```
public static boolean isMethodCalledFromRightPlace(List<String>
allowedClasses) {
    StackTraceElement[] stElements = Thread.currentThread().getStackTrace();

    /* [0] - Thread
     * [1] - LibraryProtectionUtility
     * [2] - Class calling for security check
     * [3] - Class to check for valid location
     */
    if(stElements.length >= 3) {
        return allowedClasses.contains(stElements[3].getClassName());
    }

    return false;
}
```

Kaitsemeetodi kasutuse eesmärk on järgmine: kaitsemeetod on lisatud mitmetele meetoditele teegis, mis omavad informatsiooni, mida ei tohiks välja anda kasutajale. Sisendiks sellele kaitsemeetodile antakse nimekiri klassi nimedest, kust on lubatud meetodit, kuhu see kaitsemeetod on lisatud, välja kutsuda. Selle sees analüüsitakse kaitsemeetodi välja kutsumise järjestust. Seal on tähtis kontrollida alati neljandat elementi, sest see element viitab asukohale, kust väljakutse kontrolli tehakse.

Kui väljakutsutav koht ei ole samanimelise klassiga, kui see mis on sisendi nimekirjas, siis kaitsemeetod tagastab `false` väärtuse, mis annab teada, et meetodi sisu, kus kaitsemeetod kontrolli teeb, ei tohiks tagastada, vaid tuleb anda vastav tühi väärtus.

Kui väljakutsutav koht on nimekirjas, siis on teada, et see meetodi väljakutsuja on teegi sisene ning sellisel juhul tagastab kaitsemeetod `true` väärtuse ning meetod, kus kaitsemeetod välja kutsuti, võib tagastada õige väljundi.

See kaitsemeetod ongi minu poolt loodud kaitse esmaste *reflection* vahenditega info kättesaamise kohta.

4 Ülevaade selles lõputöös esinevatest ülesannetest

Põhiline väljund käesolevale lõputööle on anda kasutajale projekt koos selle jaoks loodud teegiga. Suur osa sellest väljundist on kasutaja jaoks loodud ülesanded. Siin peatükis on välja toodud täpsem kirjeldus iga ülesande kohta, mis teegis esineb.

Ülesanded on ehitatud nii, et kasutajal on võimalik saada kõik vajalik vastavate ülesannete sisendandmetest. Neil ei ole vaja minna otsima lisa klasse või objekte, mida kasutada teegi seest. Kasutajad on teretulnud looma enda klasse ja objekte, et harjutada OOP metoodikaid, kuid see pole tehniliselt kohustuslik enamustes ülesannetes.

4.1 Objektidest arusaamine

Kuna kasutaja peab erinevate ülesannete käigus kasutama ja haldama palju eri objekte, siis on neil objektidel projektis kaasa antud dokumentatsioon JavaDoc stiilis. See tähendab, et kasutaja saab oma arenduskeskkonna abil lugeda, mida vähem selgemad meetodid ja objektid endast sisaldavad ja mida need teevad ning võimaldavad. Teegis esinevad klassid, muutujad ja meetodid on loodud niimoodi, et need oleks võimalikult oma olemust kirjeldavad, kuid siiski alati ei ole see täies mahus võimalik. Seetõttu on teatud komponendid ära dokumenteeritud täpsemal tasemel. Eriti ülesannete kirjeldused.

Samuti on kasutajale antava projektiga kaasas ka veebilehitsejas loetav ja uuritav dokumentatsioon sama JavaDoc kohta.

4.2 Soojendusülesanne

Soojendusülesanne on projektis esindatud `ExerciseWarmUp.java` faili näol. See ülesanne on mõeldud kasutajale kõige esimese ülesandena. Kui teised ülesanded pole otseselt loodud eesmärgiga, et neid peaks järjest tegema, siis see ülesanne on disainitud,

et kasutaja saaks tutvuda ja harjuda selle projekti struktuuriga ning saada parem ülevaade, kuidas seda projekti kasutada.

4.2.1 Ülesande kirjeldus ja potentsiaalne lahendus

Selles ülesandes on kasutajale ette antud neli alamülesannet. Kõik need ülesanded hõlmavad endas `Door` klassi tüüpi objekti avamist `Key` klassi tüüpi objektiga. Siin on meelega jäetud ülesande lahendused väga lihtsaks eelnevalt mainitud põhjustel.

Esimese ülesande saab lahendada kasutades neid kahte objekti omavahel. Seal pole muud vaja teha kui anda `Door` objekti `open(Key key)` meetodile `Key` objekt ette.

Teine ülesanne nõuab ühte lisasammu, kuna nüüd on `Key` objekt ära pandud `Box` objekti sisse, mille seest saab selle kätte.

Kolmandas ülesandes tuleb veel üks lisaetapp juurde. `Box` objekt on nüüd suletud ning sellel objektil tuleb enne välja kutsuda `openBox()` meetod ja siis teha sama, mis teises ülesandes.

Neljas ülesanne on ainuke, mis nõuab rohkem kui ühe lisasammu tegemist võrreldes eelmistega. Nüüd on kastis nimekiri `Key` objekte ning nende seast on vaja välja otsida õige. Seda on võimalik teha andes `Box` objekti meetodile `isItProperKey(Key key)` võtme ette, mis kontrollib `Key` objekti korrektsust.

4.3 Ülesanne: Kaustad

Ülesanne „Kaustad" on projektis esindatud `ExerciseFolder.java` faili näol. See ülesanne peaks keerukusastmelt olema üsnagi lihtne. Ülesanne on disainitud selliselt, et kasutaja saaks ühe potentsiaalse lahendusena kasutada programmeerimismetoodikat nimega rekursioon.

Rekursioon programmeerimises on kontseptsioon, kus konkreetne probleem sõltub väiksematest juhtudest samast probleemist ning selleks et seda probleemi lahendada, kasutatakse lahenduse sees sama lahendusmetoodika uuesti välja kutsumist [16] [17] . Java-s ning ka paljudes teistest programmeerimiskeeltes on võimalik seda metoodikat

rakendada mingisuguse probleemiga lahendusega tegeleva meetodi sees sellesama meetodi enda välja kutsumisega. Lihtsalt öeldes - mingi meetod kutsub iseennast välja.

4.3.1 Ülesande kirjeldus ja potentsiaalne lahendus

Selles ülesandes peab kasutaja etteantud `Folder` klassi objekti seest või selle alam `Folder` objektides üles leidma õige `ExerciseFile` klassi objeki ja tagastama selle sisu. Meetod `String exercise(Folder folder, String filename)` on antud kasutajale, kuhu ta peab oma lahenduse lõpptulemuse tagastama.

Selle ülesande üks potentsiaalne lahendus on kasutada rekursiivset lähenemist. Analüüsides iga kausta objekti sisu ja kontrollides, kas ette antud kaust omab `ExerciseFile` objekti ette antud faili nimega ning kui ei oma, siis kutsub sama meetodi välja andes nüüd sisendiks alamkausta, kui mõni eksisteerib selles kaustas, kus praegu kontroll käib. Juhul kui fail on olemas, siis tagastatakse selle faili sisu lahendusena.

4.4 Ülesanne: Šiffer

Ülesanne „Šiffer" on projektis esindatud `ExerciseCipher.java` näol. See ülesanne on ehitatud selliselt, et kasutaja peab looma uue objekti, mis implementeeriks `CipherInterface` liidest. Tehes seda, saab kasutaja harjutada oma oskusi, kuidas töötavad liidesed ning mida nendega teha saab.

4.4.1 Ülesande kirjeldus ja potentsiaalne lahendus

Selles ülesandes on vaja kasutajal luua kood, mis oskaks krüpteerida ja dekrüpteerida Caesari šifferit.

Caesari šiffer on üks enim tuntuid krüpteerimis tehnikaid [18] . See on asendus šifferi tüüpi krüpteering, kus teatud reeglite põhjal asendatakse tekst, mida soovitakse krüpteerida nende reeglite põhjal ümber töödeldud tekstiga. Caesari šifferi puhul on kasutusel krüpteerimisloogikaks tähestik erinevate tähtede ja märkidega ning krüpteerimisvõtmeks number. See number tähistab, mitu kohta tuleb iga tähe kohta tähestikus edasi liikuda. Selle numbri ja liikumise põhjal vahetatakse iga täht või märk

ette antud väärtusega tähestikus. Dekrüpteerimise puhul on tegevus vastupidine, siis liigutakse tähestikus tagasi võtme väärtuse jagu ja asendatakse krüpteeritud täht või märk dekrüpteeritud väärtusega.

Selleks, et luua Caesari šifferit, on kasutajal vaja kasutada eelnevalt mainitud `CipherInterface` liidest. Kasutades seda, on kood teadlik, et sellel objektil, mis lahendusmeetod tagastab, on olemas krüpteerimismeetod `encrypt(String text, Hashmap<String, String> parameters)` ning dekrüpteerimismeetod `decrypt(String text, Hashmap<String, String> parameters)`.

Mõlemad meetodid annavad kasutajale sisendina teksti, mida soovitakse krüpteerida või dekrüpteerida. Samuti on kasutajatele antud `HashMap<String, String> parameters` muutuja igasuguste lisaparameetrite kohta, mida võiks ühel või teisel šifferi lahendusel vaja minna. Kasutajatele on siinkohal ülesande kirjelduses välja toodud, et nende ülesannete puhul antakse neile selle *HashMap-ga* kaasa kaks parameetrit. Üks neist on *„shift"* parameeter, mis hoiab endast teksti kujul Caesari võtme numbrilist väärtust ning teine on *„alphabet"* parameeter, mis hoiab endas teksti kujul tähestiku erinevatest tähtedest, numbritest ja erimärkidest, mis on selle ülesande Caesari šifferil kasutusel.

Omades kõike seda infot ja ülesande sisendeid, on kasutajatel võimalik luua uus klass implementeerides `CipherInterface` liidest ning kirjutada selle klassi kahte vajaminevasse meetodisse ülesande lahendus funktsionaalsus.

4.5 Ülesanne: Sorteerimine

Ülesanne „Sorteerimine" on esindatud projektis `ExerciseSorting.java` faili näol. See ülesanne nõuab kasutajalt objektide analüüsi, filtreerimist ja sorteerimist. Need oskused on OOP metoodikates tihedalt kasutusel, nii et see ülesanne aitab kasutajal arendada neid oskusi.

4.5.1 Ülesande kirjeldus

Selles ülesandes on kasutajale ette antud lahendada kaks ülesannet. Mõlemad neist kasutavad `Person` klassi tüüpi objekte oma ülesande lahenduses.

Esimeses ülesandes peab kasutaja sorteerima talle ette antud nimekirja `Person` objekte. Selles klassis on ülesande kasutamiseks olemas inimese vanuse muutuja ja inimese nime muutuja. Kasutaja peab selle nimekirja etteantud isikutest sorteerima kasvavas järjekorras tähestikuliselt. See tähendab, et esmane nimekirja järjekord sõltub vanusest ning kui nimekirjas on sama vanusega isikuid mitu tükki, siis nende seast on enne inimesed, kelle nimi algab tähega, mis on tähestikus enne.

Teine ülesanne kasutab esimese ülesande probleemi, aga lisab ekstra keerukuse juurde. Selles ülesandes on vaja kasutada kolmandat `Person` klassi muutuja välja `Date lastHealthcheckDate`. See muutuja väljendab inimese objekti viimase tervise-kontrolli kuupäeva. Kui eelmises ülesandes oli vaja üks nimekiri sorteerida aja ja nime järgi, siis selles on vaja teha sama, aga nüüd tuleb see nimekiri inimesi teha neljaks eraldi nimekirjaks. Iga neist neljast nimekirjast omab teatud reeglit, mille põhjal inimesi sinna on lisatud. Kõik nelja nimekirja reeglid on seotud tervisekontrolli kuupäeva muutujaga, mida iga `Person` objekt omab. Need grupid ja nende reeglid on järgnevad:

1. Esimesse nimekirja pannakse kõik inimesed, kelle `Date lastHealthcheckDate` kuupäeva väärtus on rohkem kui aasta vana võrreldes hetke kuupäevaga. Ülesande siseselt tähendab see seda, et vastav inimene ei ole oma kindlustuse jaoks vajamineval tervisekontrollil käinud rohkem kui ühe aasta ning seetõttu määratakse ta gruppi, mis nõuab tervisekontrolli. Need inimesed lisatakse teise ülesande tagastavasse `HashMap` väärtusesse võtmetüübiga `SortingInsuranceRiskFactor.REQUIRE_CHECK`. See kindlustusriski faktori väärtus on kasutusel, kuna sellisel juhul on `HashMap` võtmeid lihtsalt hallata, kuna `enum SortingInsuranceRiskFactor` klassil on olemas ainult neli väärtust.

2. Teise nimekirja lähevad inimesed, kelle viimane tervisekontrolli kuupäev on vähem kui aasta eest ning inimene on 20-aastane või noorem. Need inimesed

lisatakse `HashMap` nimekirja `SortingInsuranceRiskFactor.LOW` võtmeväärtusega.

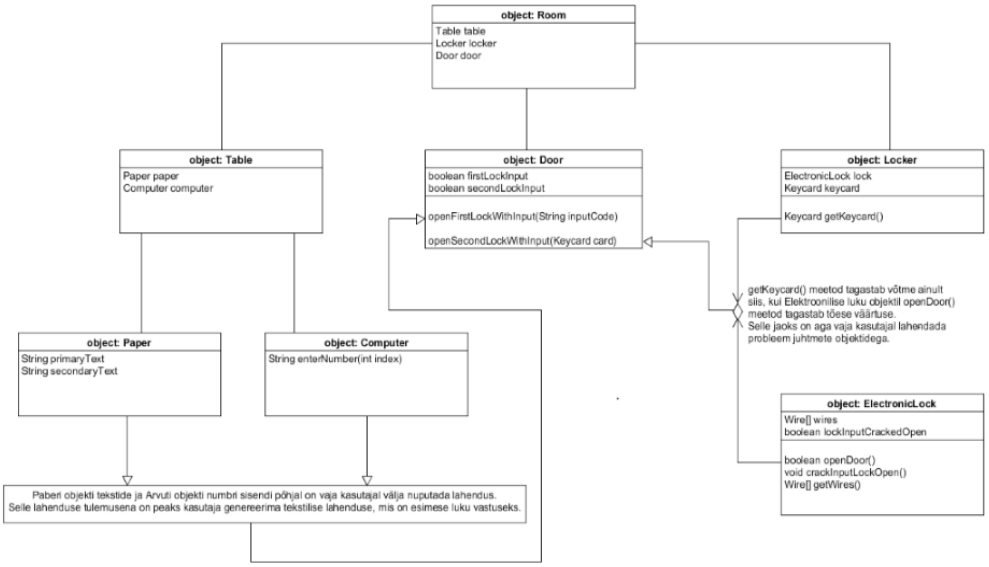
3. Kolmandasse nimekirja lähevad inimesed, kelle viimane tervisekontrolli kuupäev on vähem kui aasta eest ning inimene on vanusevahemikus 20-65. Need inimesed lisatakse `HashMap` nimekirja `SortingInsuranceRiskFactor.MEDIUM` võtmeväärtusega.

4. Neljandasse nimekirja lähevad inimesed, kelle viimane tervisekontrolli kuupäev on vähem kui aasta eest ning inimene on 65-aastane või vanem. Need inimesed lisatakse `HashMap` nimekirja `SortingInsuranceRiskFactor.HIGH` võtmeväärtusega.

4.6 Ülesanne: Pusle

Ülesanne „Pusle" on esindatud projektis `ExercisePuzzle.java` faili näol. See ülesanne on üks keerukamate hulgast, kuna siin pole probleemi püstitust otseselt välja toodud. Kasutajale antakse eesmärk ja siis on tema ülesanne läbi töödelda erinevaid objekte, mis sisendiks on antud ning nende objektide abil leida lahendus sellele ülesandele.

Selles ülesandes on mitmeid eri objekte ja nende vahelisi seoseid. Seetõttu on välja toodud järgnev joonis, mis näitab lihtsustatud kujul mitmete objektide vahelisi seoseid.



Joonis 4. Pusle ülesande objektide lihtsustatud struktuur

Siinses ülesandes on kasutajal lahendada kaks alamülesannet. Mõlemad on idee poolest samad: kasutajal on vaja kahele erinevale lukule leida õiged sisendid.

4.6.1 Esimese alamülesande kirjeldus ja potentsiaalne lahendus

Esimene lukk tahab sisendiks õiget tekstilist väärtust. Meetod selle ülesande jaoks, kuhu kasutaja peab õige vastuse tagastama, on `String codeForFirstLock(Room room)`. Selleks, et seda ülesannet ära lahendada, on vaja kasutajal esmalt uurida `Table` klass objekti. Seal klassi sees leiab kasutaja, et abstraktses mõttes laua peal on kaks asja: `Paper` klassi objekt ja `Computer` klassi objekt. Neid mõlemaid on vaja kasutada, et õige lahendus kätte saada.

Uurides neid kahte objekti saab kasutaja leida, et arvuti objektil on üks meetod `String enterNumber(int index)`. See meetod ootab numbrilist sisendit, et tagastada mingisugune tekstiline väljund. Vaadates `Paper` objekti, on sealt võimalik leida kaks

teksti muutujat, mida kasutaja saab kätte. Üks neist on mitmelauseline tekst, mis omab järgnevat väärtust:

„If you are reading this, it means you are on the right path to solving the puzzle to unlock the first lock. Now you need to find a way to use this text and alter it so you could use it to unlock the first lock. Good luck!"

Eesti keeles tõlge sellele oleks: „Kui sa loed seda, siis see tähendab et sa oled õigel teel lahendamaks pusle esimest lukku. Nüüd on sul vaja leida, kuidas kasutada seda teksti ja muuta seda selliselt, et sa saaksid avada esimese luku. Edu!"

See tekst on omab kahte eri väärtust. Esimene väärtus on see, et see on see tekst annab kasutajale vihje, kuidas esimest ülesannet lahendada. Teine väärtus on see, et see tekst nagu see ise ka ütleb, on see muutuja, mille muutmisega on võimalik esimene lukk avada.

Teine tekst paberil omab tekstilist väärtust mõnest numbrist.

Teades paberi mõlemat teksti ja arvuti numbri sisestamismeetodit, on kasutajal vaja välja mõelda lahendus. Siin kohal tulebki välja selle ülesande keerukus: kasutaja on pandud olukorda, kus otsest lahendust ei ole talle esitatud ning nagu pusled ikka, on lahendus vaja välja nuputada.

Esimese luku lahenduse loogika on järgmine: esmalt on kasutajal vaja paberi teise teksti numbrid ümber konverteerida `int` kujule. Kuna need on ainukesed numbrid, mis siin-kohal olemas on, siis võiks kasutajal tekkida seos, et neid numbreid võiks proovida sisendina arvuti objekti meetodile anda. Kui kasutaja seda teeb, siis saab ta sellelt meetodilt tagasi teksti, mis on struktureeritud selliselt, et pikast tekstist, mis on paberi peal, on välja valitud mingi hunnik tähti. Need tähed on alles jäetud ning ülejäänud tähed ja märgid on asendatud „_" ehk alakriipsu tähisega.

Kui kasutaja proovib ka teisi numbreid, mis ei ole paberil kirjas, siis ta saab sama struktuuriga väljundi, aga need on valed väljundid.

Selleks, et saada õigesti modifitseeritud paberi peamine tekst luku avamismeetodile õigesti ette antud, on vaja kasutada kõiki paberil teisel tekstil olnud numbritele

vastavaid tekste arvutis. Kõik tähed ja märgid, mis nende numbrite sisendiga välja tulid, on vaja eemaldada esialgsest paberi peamisest tekstist. See on koht, kus kasutajal tuleb see katse-eksitus-meetodika välja nuputada, aga see ongi üks pusle lahendamise osa, mida see ülesanne rakendab: katsetamine ja õigete seoste leidmine.

4.6.2 Teise alamülesande kirjeldus ja potentsiaalne lahendus

Teise luku sisendiks oodatakse `Keycard` objekti. See objekt peidab ennast `Locker` objekti sees, mida on kasutajale kättesaadav `Room` objektist, mis on ülesande sisendiks.

Kui kasutaja proovib lahenduse sisendiks anda mõnda muud, enda loodud `Keycard` objekti, siis selle jaoks on klassi konstruktoris olemas kontroll loogika, mis töötab samamoodi nagu *reflection* kaitse loogika. Kui võtmekaardi objekti tahetakse luua mujalt, kui selle õigest kohast (`Locker` klassi seest), siis seatakse ukse avamisel kontrollitav teksti väärtus valeks ning selle võtmekaardi objektiga teist luku avada ei saa.

Locker objektis, mis on toa sisend objektiga kaasas, on olemas ElectronicLock objekt. Samuti on olemas ka *getter* (meetod, mille eesmärk on tagastada muutuja) meetod võtmekaardi jaoks, kuid selle dokumentatsiooni lugedes on võimalik kasutajal teada saada, et meetod tagastab vajaminema võtmekaardi objekti ainult juhul, kui elektroonilise luku objekti meetod `boolean openDoor()` tagastab tõese väärtuse.

Selleks, et eelnimetatud meetod annaks õige väljundi, on vaja kasutajal uurida `ElectronicLock` objekti. Seal on olemas kaks meetodit, mis peaks kasutajale silma jääma. Esimene neist on `void crackInputLockOpen()` meetod, mis ütleb, et see meetod avab sulle selle elektroonilise luku objekti sisu ning annab ligipääsu luku juhtmetele. See meetod on vaja välja kutsuda enne kui minnakse teise meetodiga `Wire[] getWires()` tegelema. Kui seda pole tehtud, siis ei tagastata juhtmete massiivi.

`Wire[] getWires()` meetodi dokumentatsioon ütleb, et kui ette antud juhtmete seast lõigata katki õiged juhtmed, siis on võimalik see elektrooniline lukk avada. Juhtmete lõikamiseks on järgnevad reeglid:

- Lõika läbi kõik punased juhtmed
- Kui roheline juhe on enne punast juhet massiivis, siis ära lõika läbi punast juhet.
- Kui massiivis on rohkem kui kolm sinist juhet, siis selle asemel et lõigata läbi kõik punased juhtmed, lõika läbi hoopis kõik valged juhtmed. See reegel kehtib kõigi teiste punaste juhtme reeglitega, kui värv muutub.
- Lõika läbi täpselt üks must juhe, kui massiivis on paaritu arv valgeid juhtmeid.

See on teise luku avamise põhiline keerukus ülesande lahendamisel. See ülesanne paneb proovile, kui hästi kasutaja oskab kõik reeglite konditsioonid paika panna, et lõigataks läbi ainult õiged juhtmed, mis vaja.

Igal `Wire` objektil on olemas värvi muutuja ning muutuja, mis ütleb kas juhe on läbi lõigatud või ei. Kasutajal on vaja talle kätte saadud juhtme objektide massiivis lõigata läbi õiged juhtmed.

Kui juhtmed on õigesti lõigatud, siis `openDoor()` meetod tagastab tõese väärtuse ning kasutaja saab kätte õige võtmekaardi objekti, et teine lukk ukse objektil avada.

4.7 Ülesanne: Pokker

Ülesanne „Pokker” on projektis esindatud `ExercisePoker.java` faili näol. Siin ülesandes, on vaja tegeleda pokkeri kombinatsioonide analüüsiga. See ülesanne kasutab Texas Holdem pokkeri reeglistiku [19] .

4.7.1 Kiire ülevaade ülesandes kasutavatest pokkeri kombinatsioonidest

Ilma kogu reeglistikku detailselt üle käimata, Texas Holdem-i puhul on tegemist pokkeri variatsiooniga, kus mängijale antakse 2 kaarti ja laual on lõppseisuga 5 kaarti. Nende seitsme kaardi hulgast kasutab mängija kõige kõrgemat kombinatsiooni, mis seal esineb.

Texas Holdem-s kasutusel olevad kaardi kombinatsioonid, alatest nõrgemast, on järgmised: kõrgeim kaart, paar, kaks paari, kolmik, rida, mast, maja, nelik, mastirida, kuninglik mastirida.

Nimetatud kombinatsioonid on kasutajale antud ülesande lahendamiseks ette nähtud.

4.7.2 Ülesande kirjeldus

Selles ülesandes on kasutajale ette antud sisendiks kaks parameetrit. Esimene neist on `Hand` klassi objekt, mis omab kahte `Card` klassi objekti. Need kaardi objektid näitavad ära, millise mängukaardiga on tegu. Teine parameeter on nimekiri `Card` objektidest, mis on lauale asetatud. Selle ülesande raames on alati eelduseks, et laual on viis kaarti.

Kasutaja ülesanne on leida käe ja laual seisu põhjal kõige kõrgem olemasolev kaardi kombinatsioon ning tagastada nimekiri nendest `Card` objektidest.

Selle ülesande keerukus seisneb ülesande lahenduse mahus. Sellel ülesandel peaks teistega võrreldes olema üldjoontes kõige pikem lahendus. Tehniliselt see ei tohiks olla raskuselt ja keerukuse astmelt väga suur, kuid erinevate kaartide kombinatsioonide kontroll on see, mis teeb selle ülesande lahenduse mahukaks.

Selles ülesandes on otsest lahendust keerukas välja tuua, kuna lahendusviise, kuidas hinnata ja analüüsida kaarte, on mitmeid. Teek ise kontrollib lihtsustatult kirjeldades kasutaja lahendust järgmiselt:

Esimesena pannakse `Hand` objektis olevad kaardid ja mängulauda esindavad kaardid ühte nimekirja ning sorteeritakse kaardi tugevuse järgi. Iga kombinatsiooni jaoks on loodud meetod, mis kontrollib, kas see kombinatsioon on olemas nimekirjas. Nende kombinatsioonide kontrollmeetodite juures on ära kasutatud eelnevaid meetodeid, kui nende põhjal saab järgnevat kombinatsiooni kokku panna (näiteks maja kombinatsiooni jaoks saab ära kasutada paari ja kolmiku kontrollmeetodeid).

Kontrollimine toimub kõige tugevamast kombinatsioonist kõige nõrgemani. Selliselt saab vältida lisatööd, kuna ülesanne tahab lahendusena saada ainult kõige tugevamat kombinatsiooni. Kui leitakse kombinatsioon, siis tagastatakse selle kombinatsiooni kaardid nimekirjana ning kombinatsiooni hindamisloogika lõpetab oma töö.

5 Kokkuvõte

Käesolevas lõputöös said püstitatud järgmised eesmärgid ning saavutati järgnevad tulemused.

5.1 Eesmärgid

Selle lõputöö põhieesmärk oli luua teek, mis sisaldaks endas programmeerimis-ülesannete jaoks vajalike komponente. Teek pidi lõpptulemusena suutma anda kasutajale ülesannete jaoks kogu vajaliku ning suutma ka kasutaja lahendust kontrollida ja testida.

Valmiskujul teek pidi lisatama uude projekti, mis on mõeldud kasutajatele jagamiseks. Selle jagatava projekti eesmärk on anda kasutajatele ette esmane struktuur, kus oleks võimalik alustada teegis loodud ülesannete lahendamisega. Seal pidi olema kõik vajalik olemas, et iga ülesanne oleks lahendatav.

Kasutajale antavas projektis pidi ka olemas olema dokumentatsioon kasutajale kättesaadavate klasside kohta, mida on ülesannete jaoks vaja ning mida saab kasutada.

Lõpptulemusena pidi valmima toode, mida on võimalik anda kasutajale ning mille abil saab kasutaja hakata lahendama erinevaid ülesandeid.

Samuti pidi see lõpptoode aitama kasutajatel arendada oma Java programmeerimis-oskusi ning andma inimestele, kellel suurt JOOP kogemust pole, võimaluse sellega tutvust teha ning ennast peale nende ülesannete lahendamist mõnevõrra mugavamalt tunda objekt orienteeritud programmeerimis maailmas.

5.2 Tulemused

Selle lõputöö lõpptulemusena said kõik eesmärgid saavutatud. On olemas teek koos vaja mineva ja eelnevalt nimetatud funktsionaalsusega. Samuti sai valmis tehtud ka kasutaja jaoks loodud projekt, mille sees kasutatakse loodud teeki. Java-Doc stiilis dokumentatsioon on ka olemas, mida kasutaja saab uurida.

Teegi ehitamisel sai sinna valmis ehitatud töövoog, mis lubab lihtsalt kasutajal sisendina anda objekti ülesande klassist, kuhu ta on lisanud oma ülesande lahenduse. Seal kontrollitakse klassi vastavusi ülesannetega ning vastava seose leidmisel jooksutatakse läbi testimisloogika, mis kontrollib kasutaja sisendit. Samuti on lisatud juurde mõningane sohitegemise kaitse funktsionaalsus, mis aitab välistada tahtlikku või tahtmatut sohitegemist, kasutades selle jaoks klasse, meetodeid ja muutjaid, mis ei tohiks neile kätte saadavad olla. Kuna tegemist on lõppkasutaja käes oleva tootega, mis ei kasuta väliseid valideerimis faktoreid või muud sellelaadset loogikat, vaid on kasutajale mõeldud iseseisvaks isiklikuks kasutamiseks, siis alati on võimalus, et targemad kasutajad leivad viisi sohki teha. Kuid sellel hetkel on nende oskused juba piisavalt suured, et see projekt ei ole enam nende oskuste arendamiseks mõeldud toode ning seetõttu pole nad ka selle projekti sihtgrupp.

Kasutaja jaoks sai loodud kuus erinevat ülesannet. Need ülesanded annavad võimalusi proovile panna erinevaid tehnilisi teadmisi ja oskusi. Nende ülesannete keerukus jäi mõõdukale tasemele. See tähendab, et kasutaja, kes on suhteliselt uus Java programmeerimises ning kellel on olemas baasteadmised keelest ja kerge arusaam OOP-st, suudaks lahendada talle ette antud ülesandeid. Neil on võimalik rakendada oma eelnevaid teadmisi siinsetes ülesannetes ilma, et nad peaks teadma kõik OOP tehnilisemaid nüansse.

Siin lõputöös loodud toode on lõppkokkuvõttes kasulik toode, mis annab inimesele, kes alustab oma teed programmeerijana ning tahab omandada suuremat kogemuste pagasit JOOP teemadel, võimaluse neid oskusi arendada.

Kasutatud kirjandus

[1] Forbes, “Considering A New Job? Here Are 10 Industries In Need Of Programmers” [*Online*]. https://www.forbes.com/sites/forbestechcouncil/2017/10/18/considering-a-new-job-here-are-10-industries-in-need-of-programmers/ (18.10.2017).

[2] Eesti Töötukassa, “Raamatupidajaid ja andmesisestajaid on liiga palju, tööturg vajab lähiaastatel programmeerijaid, kokkasid ja veoautojuhte” [*Online*] https://www.tootukassa.ee/uudised/raamatupidajaid-ja-andmesisestajaid-liiga-palju-tooturg-vajab-lahiaastatel-programmeerijaid (09.07.2018).

[3] M. Kolling. "The Problem of Teaching Object-Oriented Programming, Part 1: Languages," Journal of Object-Oriented Programming, 11(8): 8-15, [*Online*] https://kar.kent.ac.uk/21879/2/the_problem_of_teaching_object-oriented_kolling_1.pdf (1999).

[4] S. Cooper, W. Dann, & R. Pausch, "Teaching objects first in introductory computer science" [*Online*] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.8151&rep=rep1&type=pdf (2003).

[5] A. Eckerdal, M. Thune, “Novice Java Programmers’ Conceptions of 'Object' and 'Class', and Variation Theory” [*Online*] http://www.it.uu.se/edu/course/homepage/datadidaktik/ht08/teaching/p89-eckerdal-thune.pdf (2005).

[6] S. Mohorovičić, Vedran Strčić, “An Overview of Computer Programming Teaching Methods” [*Online*] https://pdfs.semanticscholar.org/f5e8/a13a42f788a0169d79551a5ff90263fa2ad3.pdf (2011).

[7] Tutorials Point, “Java Tutorial” [*Online*] https://www.tutorialspoint.com/java/ (26.05.2019).

[8] Oracle, “The Java™ Tutorials” [*Online*] https://docs.oracle.com/javase/tutorial/ (26.05.2019).

[9] w3schools, “Java Tutorial” [*Online*] https://www.w3schools.com/java/ (26.05.2019).

[10] Java Point, “Java Tutorial” [*Online*] https://www.javatpoint.com/java-tutorial (26.05.2019).

[11] Code Cademy, “Learn Java” [*Online*] https://www.codecademy.com/learn/learn-java (26.05.2019).

[12] Learn Java Online, “Interactive Java Tutorial” [*Online*] http://www.learnjavaonline.org/ (26.05.2019).

[13] Oracle, “Controlling Access to Members of a Class” [*Online*] https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html (26.05.2019).

[14] G. McCluskey, “Using Java Reflection” [*Online*] https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html (1998).

[15] Java Point, “Java Reflection API” [*Online*] https://www.javatpoint.com/java-reflection (26.05.2019).

[16] Wikipedia, “Recursion (computer science)” [*Online*] https://en.wikipedia.org/wiki/Recursion_(computer_science) (26.05.2019).

[17] Java Point, “Recursion in Java” [*Online*] https://www.javatpoint.com/recursion-in-java (26.05.2019).

[18] Vikipeedia, "Caesari nihe” [*Online*] https://et.wikipedia.org/wiki/Caesari_nihe (26.05.2019).

[19] Wikipedia, “Texas hold’em” [*Online*] https://en.wikipedia.org/wiki/Texas_hold_%27em (26.05.2019).